An Algorithm for Audio Segment Fingerprinting

Devyani Hebbar

Department of Music Technology, New York University

MPATE-GE 2598: Fundamentals of Digital Signal Theory Lab

Prof. Dirk Vander Wilt

December 22, 2023

1. Abstract

This project aims to write a scaled-down version of an audio fingerprinting algorithm. On conducting a literature survey of existing methods for this task, I chose the method used in the mobile application Shazam that has been detailed in [2]. Two functions are created in this project, one for extracting spectrogram peaks and one for extracting combinatorial hashes. A result analysis is conducted by comparing the hashes generated for a segment of a song against hashes generated for the entire song, as well hashes generated for two other (different) songs. There is a clear result for which song is the fingerprint.

2. Introduction

An audio fingerprint is a “signature” that contains information about an audio recording. Audio fingerprinting is the process of identifying an unlabeled audio recording by its fingerprint. An audio fingerprinting algorithm must be able to listen to an input audio file, survey a database of several audio files, and then correctly match the input audio file to one of the audio files in the database. This project is an attempt at creating a scaled-down version of an audio fingerprinting algorithm. The aim of the project is to be able to write an algorithm that creates a fingerprint for one segment of a song and then correctly identifies that segment when presented with the whole song.

The task of Audio Fingerprinting is one of the most fundamental topics in Music Information Retrieval. The ability of machines to identify and label music in real time highly increases the opportunity to discover new music for musicians and listeners alike. As music technologists, the end goal of our projects must be to serve either musicians or listeners or both. Hence, it is beneficial for us to understand how such an algorithm works and the computational principles upon which it is built.

In this project, I have first imported the audio file of the song I want to fingerprint. The audio file is then analyzed by applying the Short-Time Fourier Transform (STFT) and various functions are created to extract information about its frequency spectrum from its STFT. This information is stored in a data frame. A segment of the song is then imported as a separate audio file, and the same functions are applied to this segment and its corresponding information is stored in another data frame. The two data frames are then compared. This process is then repeated for a segment of a different song.

3. Literature Review

Audio fingerprinting is a content-based identification technology. It is different from audio watermarking, which is the process of embedding an arbitrary message into the audio without changing the perception of it. In audio fingerprinting, this "message" is automatically obtained from the perceptually important parts of the audio. Some qualities that an audio fingerprinting algorithm must have include invariance to distortion, compactness and being easily computable. The algorithm must be able to label audio files even with added distortions (such as ambient noise). The fingerprints must also be easily extractable and take up as little storage space as possible. One of the most commercially successful mobile applications that performs the task of audio fingerprinting is Shazam. The first step in the Shazam algorithm is to convert the audio file into a spectrogram, which is a visual representation of frequency intensities over time. The algorithm then chooses certain frequency values known as "peaks" from the spectrogram. The peaks are frequency values that have the highest intensity. These values are chosen because high frequencies are more robust to noise as compared to low frequencies. The amplitude component of the spectrogram is then eliminated and this visualization is called a constellation map. It is possible to search for matches using just constellation maps, however, this would result in extremely long matching times due to the fact that the map is sparse. Therefore, Shazam uses a technique called Combinatorial Hashing to find matches. Every point on the map is considered an anchor point, and a target zone is decided for each anchor point. The other points in this zone are called considered target points with reference to anchor points. Every anchor point is paired with a target point and these pairs are called combinatorial hashes. The hashes are essentially pairs of frequency peak values along with the interval between the timings at which they occur in the song. The logic behind using these hashes is that it is likely for two distinct songs to share the same frequency values, but it is less likely for two distinct songs to have the same frequency values that are the exact same time interval apart. Therefore, combinatorial hashes allow the algorithm to very quickly identify fingerprint matches.

4. Goals and Methodology

The audio recording to be fingerprinted (a wav file of the song "Crazy" by Britney Spears) was first imported and stored in a variable 'y' along with its sample rate in variable 'sr' using the 'librosa.load' function. Next, the Short-Time Fourier Transform was applied to the audio signal using the 'librosa.stft' function and converted to a logarithmic scale using 'librosa.amplitude_to_db'. This was visualized as a spectrogram as seen in figure 1.

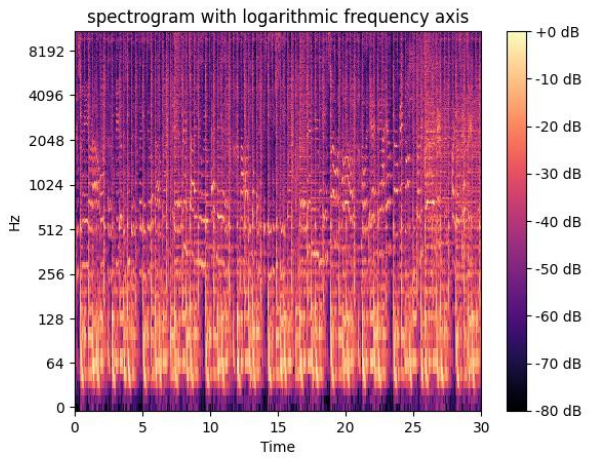


Figure 1.

Once audio file was converted to this spectrogram, a function was created to extract its peaks. The function is called 'get_peaks' and requires two input parameters: an STFT array and a threshold value. The function first creates a normal distribution of frequency intensities from the STFT, and creates a z-score for each value. This is a measure of how far away from the mean each value is in terms of the standard deviation of the distribution. The z-scores a are stored in a matrix. This matrix is then filtered by keeping only those values that are greater than the threshold specified as an input parameter to the function. The values in this filtered matrix are then arranged as an evenly spaced array using the 'numpy.arange' function and grouped into various regions. Finally, the maximum value in each region is found using the 'extract_region_maximus' function from the scipy library. These maximum-intensity frequency values are then stored in a one dimensional list called 'peaks'. This completes the 'get_peaks' function. On applying this function to the audio signal at hand, the peaks were stored and then visualized as seen in figure 2.

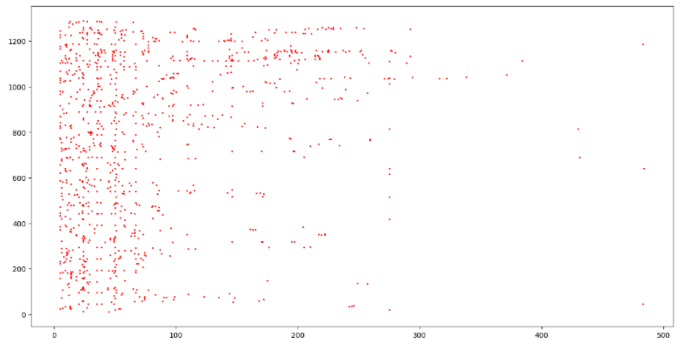
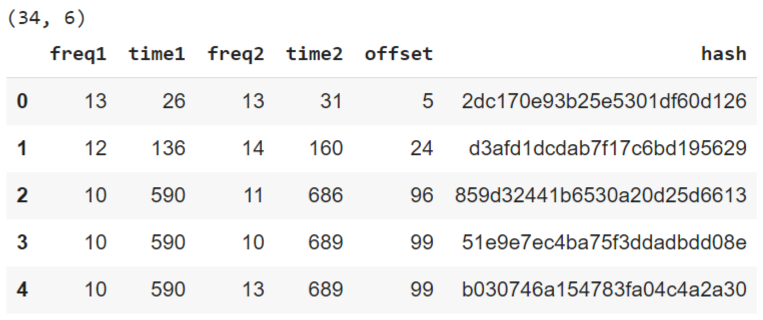


Figure 2

The next step was to create combinatorial hashes for the obtained peaks. A function called 'get_hashes' was written for this. The function requires two input parameters: peaks (the one dimensional list of peaks obtained from the ‘get_peaks’ function) and peak_combination, which is a singular integer value. This function uses a nested for loop to iterate over each peak frequency value and arrange them in pairs and calculate the difference between the times at which they occur. The ‘hashlib.sha1’ function is then used to generate a unique ID comprising of letters and digits for each frequency pair and its corresponding time interval. This completes the ‘get_hashes’ function. The ‘get_hashes’ function is then applied to only a segment of the song (only a subset of the peaks). The generated combinatorial hashes are then stored in a pandas data frame as seen in figure 3. They are also visualized as seen in figure 4.



```
(34, 6)
```

	freq1	time1	freq2	time2	offset	hash
0	13	26	13	31	5	2dc170e93b25e5301df60d126
1	12	136	14	160	24	d3afd1dcdab7f17c6bd195629
2	10	590	11	686	96	859d32441b6530a20d25d6613
3	10	590	10	689	99	51e9e7ec4ba75f3ddadbdd08e
4	10	590	13	689	99	b030746a154783fa04c4a2a30

Figure 3

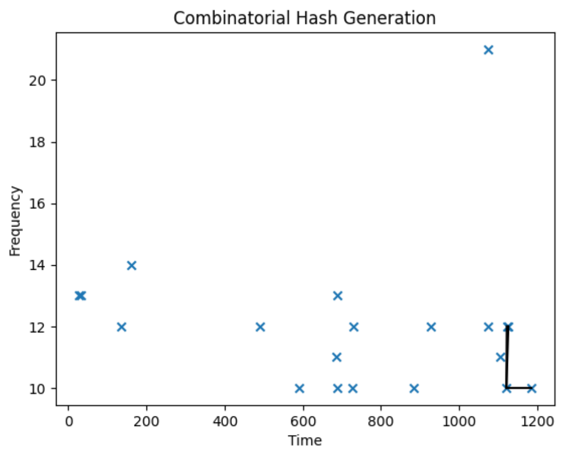
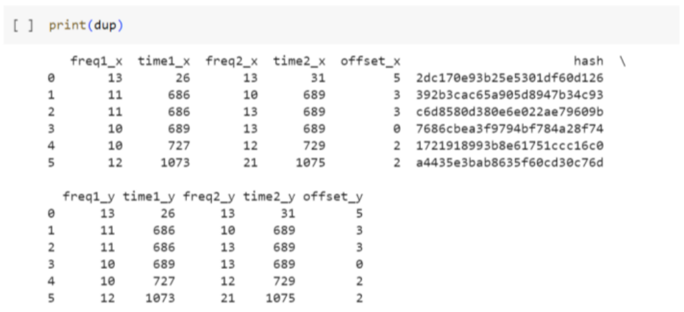


Figure 4

5. Results and Analysis

To test the functions, I imported the same wav file and applied the ‘get_peaks’ and ‘get_hashes’ functions to its STFT. Only this time, I obtained hashes for the entire song and not just a segment of it. These hashes were stored in a separate pandas dataframe. The two dataframes were then compared using the pandas.merge function to display all the rows where the datapoints in the ‘hash’ column are identical. This resulted in six rows as seen in figure 5.



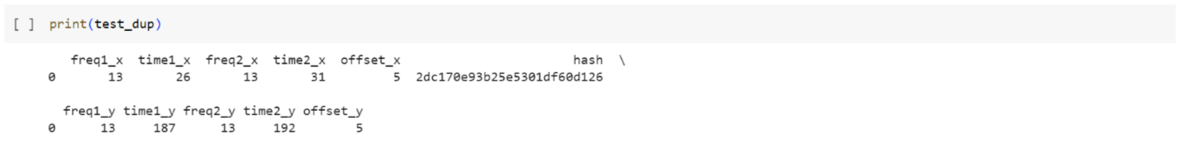
```
[ ] print(dup)
```

```
   freq1_x  time1_x  freq2_x  time2_x  offset_x                      hash  \
0       13       26       13       31         5  2dc170e93b25e5301df60d126
1       11      686       10      689         3  392b3cac65a905d8947b34c93
2       11      686       13      689         3  c6d8580d380e6e022ae79609b
3       10      689       13      689         0  7686cbea3f9794bf784a28f74
4       10      727       12      729         2  1721918993b8e61751ccc16c0
5       12     1073       21     1075         2  a4435e3bab8635f60cd30c76d

  freq1_y time1_y freq2_y time2_y offset_y
0      13      26      13      31        5
1      11     686      10     689        3
2      11     686      13     689        3
3      10     689      13     689        0
4      10     727      12     729        2
5      12    1073      21    1075        2
```

Figure 5

This means that there are six instances within the chosen segment of the song where two specific frequency peaks are the same time interval apart as in the full song. The same test was also performed on two other songs. The first song had only one hash in common with the original segment while the second one had no hashes in common.

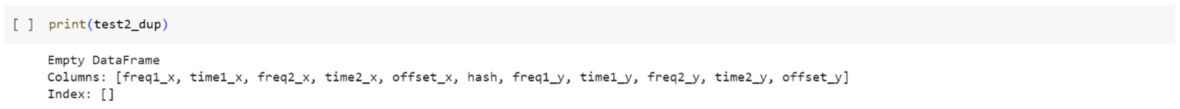


```
[ ] print(test_dup)
```

```
   freq1_x  time1_x  freq2_x  time2_x  offset_x                      hash  \
0       13       26       13       31         5  2dc170e93b25e5301df60d126

  freq1_y time1_y freq2_y time2_y offset_y
0      13     187      13     192        5
```

Figure 6



```
[ ]  print(test2_dup)

     Empty DataFrame
     Columns: [freq1_x, time1_x, freq2_x, time2_x, offset_x, hash, freq1_y, time1_y, freq2_y, time2_y, offset_y]
     Index: []
```

Figure 7

6. Conclusions

Since a higher number of common combinatorial hashes means a higher chance of there being a match, this implies a high likelihood that the song whose hashes are displayed in figure 5 is a fingerprint of the original segment. On the other hand, it can be inferred that the songs whose hashes are displayed in figure 6 and figure 7 are not fingerprints of the original segment.

7. Future Work

This project can further be developed by creating a Machine Learning model that would be able to compare combinatorial hashes between audio recordings and determine whether they are fingerprints of each other or not. In addition, the dataset that the model is trained on can be expanded to multiple songs rather than just one.

References

Cano, P., Batlle, E. (2005). *A Review of Audio Fingerprinting.* Journal of VLSI Signal Processing 41, p. 271–284, 2005.

Froitzheim, S. (2017). *A Short Introduction to Audio Fingerprinting with a Focus on Shazam.*

https://williamsantos.me/posts/2023/spectrogram-peak-detection-with-scipy/, n.d.

https://willdrevo.com/fingerprinting-and-audio-recognition-with-python/, n.d.

https://medium.com/swlh/understanding-audio-fingerprinting-b39682aa3b5f, n.d.